

22

QUE LE FORTH SOIT AVEC VOUS

MARS 1986

F-BASE I

Base de données en FORTH

EDITORIAL

De nombreux programmes et des sujets variés dans cette mouture printannière de JEDI, avecun nouveau venu: APL.

Peut-être nombreux sont ceux qui ne se sentent que peu ou point concerné. Et pourtant, nos sources d'informations nous ont signalé que ce langage a été implanté sur un système familial très répandu, mais n'a point été commercialisé car il aurait été nécessaire de faire l'acquisition d'une extension mémoire d'au moins 250 K de capacité dont le prix aurait été nettement supérieur à celui de l'unité centrale. Peut-être n'est ce que partie remise. En effet, si l'on considère que la micro d'aujourd'hui correspond à l'informatique d'hier, du point de vue des performances et des capacités mémoire, il semble raisonnable de penser que ce glissement ira en s'accentuant. Déjà maintenant, une capacité de 128 K est monnaie courante, alors que les pionniers de la micro rêvaient d'extensions 2K ou 4K.

L'argument classique et communément répandu laisse croire qu'avec une capacité accrue de la mémoire des systèmes, il ne sera plus nécessaire de faire appel aux compilateurs pour gagner de la place; que les micros tourneront plus vite et que le BASIC ira en s'améliorant.

Or, on constate de plus en plus sur certains BASIC, l'acquisition de commandes semblables à celles existant sur d'autres langages. Pour mémoire, voici quelques cas types: les BASICs du QL SINCLAIR ou du BBC ACORN possèdent la notion de procédure et de variables locales (presque du LSE ou du PASCAL interprété ...); le BASIC 3X du HECTOR HRX qui peut utiliser des commandes FORTH, etc... Mais plus fort encore, je cite les nombreux essais de langages écrits à partir d'autres langages. On trouve en abondance des Tiny-LISP ou Tiny-PASCAL écrits en FORTH, des FORTH écrits en C, des PROLOG et des LISP écrits en PASCAL, des LOGO écrits en BASIC ou en FORTH. Alors pourquoi ne pas imaginer un FORTH écrit en C à partir duquel est développé un PASCAL qui permettrait de développer un PROLOG !!

Mais sans délirer, nous vous promettons de vous étonner, et pas plus tard que le prochain numéro de JEDI où il sera question de FORTHLOG, un langage écrit en FORTH, permettant de réaliser une grande variété de systèmes experts (tant pis pour ceux qui ne se réabonnent pas ..).

SOMMAIRE

ASSOCIATION JEDI 8, rue Poirier de Narçay 75014 PARIS
Tel: (1) 45.42.88.90 (de 10h à 18h)

Par Jérome LEONIS

DEMYSTIFIER APL
... UN LANGAGE QU'IL FAUT CONNAITRE

APL est un langage interprété bien implanté aux Etats-Unis et en Europe Occidentale, sur des gros et moyens systèmes, et à présent sur des micro-ordinateurs.

Si le FORTRAN, le PASCAL sont surtout utilisés dans des applications scientifiques, et le COBOL dans des applications de gestion, les Systèmes APL s'adressent à tous les types de travaux, grâce à un environnement aujourd'hui très diversifié.

En effet, chez IBM France pour les avoir utilisés, j'ai travaillé sur des produits bien utiles pour le Marketing et le Management comme APE permettant la gestion d'écrans, DCF le traitement de textes, GRAPHPAK et APGS pour éditer des courbes et des tableaux de données, ADI pour interroger les fichiers. De même, ont été conçus sur des systèmes APL, des logiciels de CAO, (Conception Assistée par Ordinateur), des applications industrielles et scientifiques, etc...

Si APL est très apprécié par ceux qui le connaissent, et qui savent qu'à l'aide de quelques opérateurs dans une ligne APL, on peut accomplir un travail qui nécessiterait plusieurs lignes d'un autre langage, il est grandement boudé par ceux qui considèrent que la programmation d'APL est un tissu d'alchimie dont la maintenance est quasi impossible.

A ces derniers, je montrerai après avoir présenté les concepts de base du langage, des moyens de concevoir des applications "lisibles" en APL.

1° Tout d'abord, il faut savoir que l'APL repose sur le concept de zone de travail active:

Si l'on veut concevoir un programme, on va définir pour cela une zone de travail qui contiendra tout ce que l'on tapera au terminal: on définira ainsi des variables et des fonctions qui pourront être mémorisées lorsque l'on sauvegardera cette zone de travail sur disque.

Contrairement aux autres langages, APL n'utilise pas les mots clé que chacun connait comme FOR, PRINT ou IF. Il n'utilise que des symboles appelés "opérateurs", et dont les fonctions sont beaucoup plus riches.

Quant aux variables, il n'est pas nécessaire de les déclarer, et elles peuvent être redéfinies constamment dans le programme.

Un autre facilité: APL est un langage interactif, ce qui permet par exemple pendant la mise au point de corriger des erreurs dans les fonctions au fur et à mesure de l'exécution du programme.

2° Comment construire une fonction APL:

Prenons un premier exemple: soit un vecteur V (qui correspond à un tableau de données numériques dans un autre langage), on écrit:

```
V ← 2 3 8
```

Pour définir Z, la somme de ces valeurs, il suffit d'écrire:

```
Z ← +/ V
```

qui se lit réduction par somme du vecteur V. On peut alors définir une fonction du nom de SOMME qui ferait l'opération de réduction. De façon plus simple et plus "lisible" on écrira:

```
Z←SOMME V
```

2 caractères APL pour l'exécution !!

```
      ∇Z←SOMME V
[1]   Z←+/V
[2]   ∇
      SOMME 23 45 78 95
241
      S←SOMME 23 45 78 95
      S
241
      Z←SOMME 23 45 78 95
      Z
241
```

La fonction SOMME est alors un opérateur particulier qui pourra être utilisé dans un programme autant de fois que nécessaire. On construira donc des fonctions de plus en plus complexes, qui elles-même appeleront des fonctions plus simples, et ainsi de suite...

L'exemple suivant nous montre l'exécution d'une fonction: réduction d'un tableau de prénoms.

A partir du tableau P:

```
|JEAN|
|PAUL|
|JEAN|
```

on veut définir le tableau Z:

```
|JEAN|
|PAUL|
```

qui ne contient chaque prénom qu'une seule fois.

Une seule ligne de programmation en APL a suffit pour transformer la matrice P.

Il faut noter que la fonction REDUIT permet de transformer une matrice de dimension quelconque.

```
    ∇ Z←REDUIT P
[1]   Z←(∨⌿<\P∧.=⍉P)⌿P
    ∇

[1]   Z←(∨⌿<\P∧.=⍉P)⌿P

JEAN
PAUL
JEAN

[1]   Z←(∨⌿<\P∧.=⍉P)⌿P

JEAN
PAUL
JEAN

[1]   Z←(∨⌿<\P∧.=⍉P)⌿P

JPJ
EAE
AUA
NLN

[1]   Z←(∨⌿<\P∧.=⍉P)⌿P

 1 0 1
 0 1 0
 1 0 1

[1]   Z←(∨⌿<\P∧.=⍉P)⌿P

 1 0 0
 0 1 0
 1 0 0

[1]   Z←(∨⌿<\P∧.=⍉P)⌿P

1 1 0

[1]   Z←(∨⌿<\P∧.=⍉P)⌿P

JEAN
PAUL
```

Suite page 5

XI STRUCTURE DES PROGRAMMES

Comme nous l'avons dit dans le résumé précédent une routine n'est, ni plus ni moins, qu'une collection de lignes de commandes. Ces routines sont stockées sur disque par l'interpréteur MUMPS. Elles sont considérées comme entité unique. Nous ne vous rappelerons pas la convention concernant la première ligne d'une routine.

La dernière phase nécessaire, pour la création d'une solution informatique, consiste à lier un certain nombre de routines entre elles. Cet ensemble homogène est réalisé en utilisant les commandes DO/QUIT. L'utilisation des commandes DO/QUIT permet de créer une chaîne de taches modulaires. Chacune des taches résoudra un probleme precis et limité. La commande GOTO NOM DE ROUTINE est utilisée essentiellement pour transférer le contrôle d'exécution à un autre programme.

A) GOTO et DO/QUIT utilisées pour les routines

Nous pouvons dire de suite que deux possibilités existent pour accéder aux routines préalablement stockées. Cet appel peut être fait en utilisant soit la commande GOTO, soit la commande DO. Ces deux commandes ont été étudiées dans le chapitre précédent. L'appel a une routine stockée est réalisé en précisant à la machine qu'elle doit chercher dans la mémoire de masse. La machine comprendra cette ordre, si on fait précéder le nom de la routine d'un accent circonflexe. Exemple :

```
GOTO ^ROUTINE1
```

Cet ordre peut être assimilé à la commande GOTO ETIQUETTE de la routine déjà chargée en mémoire. Notez d'ores et déjà que l'interpréteur MUMPS, à la rencontre d'un accent circonflexe devant le nom d'une routine ou d'une variable, cherchera sur disque. Lorsqu'on cite, simplement, le nom d'une routine dans une commande de transfert d'exécution (GOTO,DO), l'exécution commence à la première ligne de la routine appelée. Imaginons, pour exemple, (bien que ce ne soit pas la méthode de programmation en MUMPS) l'enchainement de routines suivant pour la gestion d'un compte en banque personnel :

```
DEBUT    ;PRG. DE DEBUT Y.L.G. 1/10/84  2/10/84
         ;identification de la banque en cause
...      ...
         G ^CREDIT


CREDIT   ;PRG. DE M.A.J BANQUE CREDIT Y.L.G. 1/10/84  1/10/84
         ;remise de cheque(s)/impression bordereau
...      ...
         G ^DEBIT


DEBIT    ;PRG. DE M.AJ. BANQUE DEBIT Y.L.G. 2/10/84  2/10/84
         ;comptabilisation des retraits
...      ....
         G ^SOLDE

SOLDE    ;PRG. DE SOLDE DE BANQUE Y.L.G. 2/10/84  2/10/84
         ;calcul et impression du solde d'une banque
...      ...
AUTRE    R !,"avez-vous d'autres entrees a effectuer (O/N) :",R
         G FIN:R="N",^DEBUT:R="O",AUTRE
FIN      Q  ;fin de traitement
```

→

Dans l'exemple ci-dessus, nous avons utilisé une méthode de programmation qu'un spécialiste appellera "linéaire" et "non structurée". Nous pouvons, bien entendu, programmer d'une facon différente. Ceci est obtenu à l'aide de l'instruction DO/QUIT. Reprenons l'exemple ci-dessus en y apportant quelques modifications :

```
DEBUT     ;PRG. DE DEBUT Y.L.G. 1/10/84  2/10/84
          ;identification de la banque en cause
...       ...
          S R="O"
          I R'="O" G FIN^SOLDE
          D ^CREDIT
          D ^DEBIT
          D ^SOLDE
          G DEBUT+2


CREDIT    ;PRG. DE M.A.J BANQUE CREDIT Y.L.G. 1/10/84  1/10/84
          ;remise de cheque(s)/impression bordereau
...       ...
          Q


DEBIT     ;PRG. DE M.AJ. BANQUE DEBIT Y.L.G. 2/10/84  2/10/84
          ;comptabilisation des retraits
...       ....
          Q


SOLDE     ;PRG. DE SOLDE DE BANQUE Y.L.G. 2/10/84  2/10/84
          ;calcul et impression du solde d'une banque
...       ...
AUTRE     R !,"avez-vous d'autres entrees a effectuer (O/N) :",R
          Q:"O/N"[R  G AUTRE
FIN       Q  ;fin de traitement
```

Il existe encore une autre manière d'écrire ce petit exercice. Nous commenterons, à la fin de ce dernier exemple, la nouvelle forme d'utilisation de la commande GOTO (G FIN^SOLDE).

```
DEBUT     ;PRG. DE DEBUT Y.L.G. 1/10/84  2/10/84
          ;identification de la banque en cause
...       ...
          S R="O"
TANTQUE   I R="O" D ^CREDIT,^DEBIT,^SOLDE G TANTQUE
FIN       Q  ;fin de traitement


CREDIT    ;PRG. DE M.A.J BANQUE CREDIT Y.L.G. 1/10/84  1/10/84
          ;remise de cheque(s)/impression bordereau
...       ...
          Q


DEBIT     ;PRG. DE M.AJ. BANQUE DEBIT Y.L.G. 2/10/84  2/10/84
          ;comptabilisation des retraits
...       ....
          Q


SOLDE     ;PRG. DE SOLDE DE BANQUE Y.L.G. 2/10/84  2/10/84
          ;calcul et impression du solde d'une banque
...       ...
AUTRE     R !,"avez-vous d'autres entrees a effectuer (O/N) :",R
          Q:"O/N"[R  G AUTRE
```

→

Non seulement il est possible d'appeler une routine avec l'ordre DO pour exécuter une sous-tache, mais aussi de passer le contrôle d'exécution à l'aide de la commande GOTO. De plus, MUMPS permet de spécifier un point d'entrée (une etiquette) dans les routines appelées. Nous avons illustré ce phénomène avec la commande GOTO FIN^SOLDE. Cette commande passe le contrôle d'exécution à la ligne FIN de la routine SOLDE.

RESUME

Au point où nous sommes parvenu dans notre apprentissage, nous avons étudié comment écrire un ensemble de routines permettant de résoudre un problème complet. Dans les chapitres suivants, nous étudierons comment organiser les informations véhiculées par MUMPS, aussi bien pour les informations temporaires (stockées dans les variables "mémoire"), que pour les informations permanentes stockées dans ce qu'on a l'habitude d'appeler des fichiers (variables globales).

suite de la page 2.

Le lecteur peut être dérouté par la logique qui permet d'arriver au résultat. Pas de récurrence, pas d'itération, mais simplement une succession d'opérations sur des matrices et des vecteurs.

Avec de l'expérience et en faisant appel à vos notions d'algèbre linéaire, vous vous apercevrez qu'il n'est pas si difficile de s'adapter à cette nouvelle forme de raisonnement.

A présent analysez le nombre d'instructions COBOL qu'il faudrait écrire pour réduire ce tableau de prénoms...vous commencez à avoir le virus APL.

Vous voulez ajouter un prénom à la liste P, il suffit de construire une fonction "AJOUTE", et la phrase en APL s'écrit:

```
P <- P AJOUTE "LUC"
```

Vous pouvez de la même façon écrire:

```
      P←P AJOUTE 'JACQUES'
      P
JEAN
PAUL
JEAN
JACQUES
      P CHERCHE 'PAUL'
2
      P CHERCHE 'FERNANDE'
0
      P←P EFFACE 'JACQUES'
      P
JEAN
PAUL
JEAN
      Z←REDUIT P
      Z
JEAN
PAUL
```

Il est donc possible d'écrire des programmes en APL qui utilisent des fonctions ayant des noms mnémotechniques.

L'alchimie d'APL ne se trouverait que dans des fonctions que j'appelerai "utilitaires" (REDUIT AJOUTE, EFFACE ...).

Enfin APL permettant de former des "GROUPES" de fonctions, on peut ainsi construire des applications facilement maintenables.

A présent, si vous êtes convaincu par la souplesse d'utilisation de l'APL, et si vous admettez qu'il existe des méthodes pour rendre votre programmation "lisible", vous connaissez vous aussi la joie d'être un APListe.

J.LEONIS (E.S.I. MONTREUIL)

REVUE DE PRESSE

Revue de presse partielle devrai-je écrire, car il n'y est question que des revues se consacrant partiellement ou totalement au FORTH:

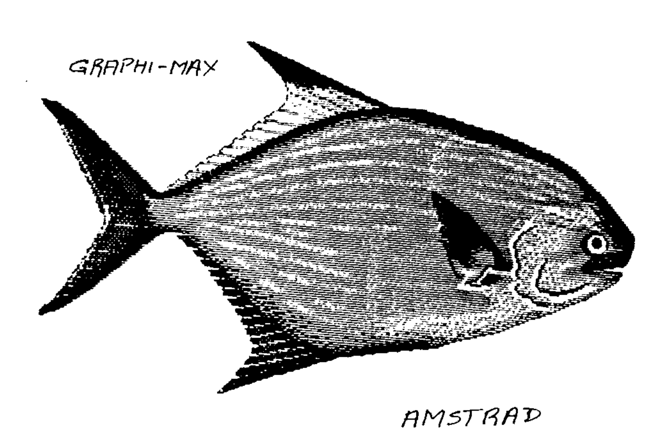
MICRO-REVUE: le support bimestriel de l'association PPC-T TOULOUSE propose dans son dernier numéro une implantation du langage BLAISE qui est réalisée ... en FORTH !! En fait, il s'agit tout simplement d'un "Tiny-PASCAL", mais l'abondance des explications (l'article est en 4 parties) montre comment est défini un interpréteur/compilateur BLAISE. Vous y trouverez également un article sur une expérience d'apprentissage à la programmation en FORTH en milieu scolaire.

TEOPHILE: une revue verticale consacrée aux micros THOMSON sort maintenant en kiosque. Dès le numéro deux, une rubrique sera consacrée régulièrement au langage FORTH.

AMSTRAD USER: revue britannique publie de temps en temps des programmes en FORTH. A noter: dès les premiers numéros, un article d'initiation vous introduisait dans les méandres de ce langage captivant alors que le logiciel FORTH n'était pas encore disponible dans le commerce.

MICRO-SYSTEMES: une des rares revues françaises non dédiée à un matériel qui ne ressemble pas à un catalogue; dans le numéro de mars 86, vous saurez tout sur le coprocesseur numérique 8087; à noter, dans le cahier des programmes, une application de cryptage de données écrite en PASCAL et son équivalent BASIC

Adresse de PPC-T: 77, rue du Cagire, 31100 TOULOUSE.



BLOC # 11

```
( ASSEMBLEUR FORTH pour 8086                          page 1/1     )
VOCABULARY ASSEMBLER IMMEDIATE        ASSEMBLER DEFINITIONS   BIN

: END-CODE ↑COMPILE§ SMUDGE ;
: NEXT, 11001011 C, ;

1000 VARIABLE REG                         1000 VARIABLE R/M
0 VARIABLE D          0 VARIABLE S        0 VARIABLE W
0 VARIABLE MOD                            0 VARIABLE DISP

: PTR 0 D ! ;                             : SEG REG ! 1 W ! 1 S ! 0 D ! ;
: REGS 10 D ! DUP REG !
                 R/M à       1000 = IF R/M ! 11 MOD ! ELSE DROP
                                                          THEN ;
: REGW 1 W ! REGS ;                     : REGB 0 W ! REGS ;
: INIT 1000 DUP REG ! R/M ! 0 D ! 0 W ! 0 MOD ! 0 S ! ;    -->
```

BLOC # 12

```
( Assembleur FORTH pour 8086                        page 2/13    )

: ES    0 SEG  ; : CS    1 SEG  ; : SS   10 SEG  ; : DS   11 SEG  ;

: AL    0 REGB ; : CL    1 REGB ; : DL   10 REGB ; : BL   11 REGB ;
: AH 100 REGB ; : CH 101 REGB ; : DH 110 REGB ; : BH 111 REGB ;

: AX    0 REGW ; : CX    1 REGW ; : DX   10 REGW ; : BX   11 REGW ;
: SP 100 REGW ; : BP 101 REGW ; : SI 110 REGW ; : DI 111 REGW ;

: W, 1 W ! ; : B, 0 W ! ; : ) 110 R/M ! 0 MOD ! DISP ! PTR ;
: 0DS 0 DISP ! PTR 0 MOD ! ;   : DS! DISP ! PTR 10 MOD ! ;
: ↑BX+SI§  0 R/M ! 0DS ;   : +BX+SI§  0 R/M ! 10 MOD ! DS! ;
: ↑BX+DI§  1 R/M ! 0DS ;   : +BX+DI§  1 R/M ! 10 MOD ! DS! ;
: ↑BP+SI§ 10 R/M ! 0DS ;   : +BP+SI§ 10 R/M ! 10 MOD ! DS! ;
: ↑BP+DI§ 11 R/M ! 0DS ;   : +BP+DI§ 11 R/M ! 10 MOD ! DS! ; -->
```

BLOC # 13

```
( Assembleur FORTH pour 8086                          page 3/13    )

: ↑SI§ 100 R/M ! 0DS ;      : +SI§ 100 R/M ! 10 MOD ! DS! ;
: ↑DI§ 101 R/M ! 0DS ;      : +DI§ 101 R/M ! 10 MOD ! DS! ;
                            : +BP§ 110 R/M ! 10 MOD ! DS! ;
: ↑BX§ 111 R/M ! 0DS ;      : +BX§ 111 R/M ! 10 MOD ! DS! ;

: CW W à IF , ELSE C, THEN ;

: BYTE2 MOD à 10 = DISP à 100000000 U< * IF 1 MOD ! THEN
        MOD à   11 SHL
        REG à + 11 SHL R/M à + C,
              10 MOD à =    IF DISP à ,  THEN
               1 MOD à =    IF DISP à C, THEN
   110 R/M à = 0 MOD à = * IF DISP à , THEN INIT ;
-->
```

BLOC # 14

```
( Assembleur FORTH pour 8086                             page 4/14 )

: MVI MOD à 11 = IF 10110 W à +      11 SHL REG à + C,
                 ELSE 11000110 W à + C, 0 REG ! BYTE2
                 THEN CW INIT ;

: MOV S à IF 10001100 D à  0= IF 10 ELSE 0 THEN  + C, BYTE2
          ELSE W à D à + MOD à 0= R/M à 110 = * REG à 0= *
                         IF   10100000 + C, DISP à ,

                         ELSE 10001000 + C, BYTE2
                         THEN
          THEN INIT ;

-->
```

BLOC # 15

```
( Assembleur FORTH pour 8086                         page 5/13   )

 : PUSH S à IF REG à 11 SHL 110 + C,
            ELSE MOD à 11 = IF 1010000 REG à + C,
                           ELSE -1 C, 110 REG ! BYTE2
                            THEN
            THEN INIT ;

 : POP S à IF REG  à 11 SHL 111 + C,
           ELSE MOD à 11 = IF 1011000 REG à + C,
                          ELSE 10001111 C, 110 REG ! BYTE2
                           THEN
           THEN INIT ;

 : XCHG 10000110 W à + C, BYTE2 ;
 -->
```

BLOC # 16

```
( Assembleur FORTH pour 8086                            page 6/13 )

 : XLAT 11010111 C, ;
 : LEA  10001101 C, BYTE2 ;
 : LDS  11000101 C, BYTE2 ;
 : LES  11000100 C, BYTE2 ;

 : SW 10000000 OVER 100000000 U< W à * IF 11 + C, 1 S !
                                  ELSE W à + C, 0 S !
          THEN BYTE2
         S à 0= W à *          IF  , ELSE C, THEN INIT ;

 : ADD D à W à + C, BYTE2 ;
 : ADI REG à 0= MOD à 11 = * IF 100 W à + C, CW
                            ELSE 0 REG ! SW
                            THEN INIT ; -->
```

BLOC # 17

```
( Assembleur FORTH pour 8086                          page 7/13)

 : ADC 10000 D à + W à + C, BYTE2 ;

 : ACI REG à 0= MOD à 11 = * IF 10100 W à + C, CW
                            ELSE 10 REG ! SW
                            THEN INIT ;
 : INC 11 MOD à = W à * IF 1000000 REG à + C,
                  ELSE 0 REG ! 11111110 W à + C, BYTE2 THEN INIT ;

 : SUB 101000 D à + W à + C, BYTE2 ;

 : SUI REG à 0= MOD à 11 = * IF 101100 W à + C, CW
                           ELSE 101 REG ! SW
                           THEN INIT ;
 -->
```

BLOC # 18

```
( Assembleur FORTH pour 8086                          page 8/13   )

 : SBB 11000 D à + W à + C, BYTE2 ;
 : SBI REG à 0= MOD à 11 =`* IF 11100 W à + C, CW
                            ELSE 11 REG ! SW
                            THEN INIT ;

 : DEC MOD à 11 = W à * IF 1001000 REG à + C,
                  ELSE 1 REG ! 11111110 W à + C, BYTE2
                  THEN INIT ;
 : NEG 11110110 W à + C, 11 REG ! ;
 : CMP 111000 D à + W à + C, BYTE2 INIT ;
 : CPI REG à 0= MOD à 11 = * IF 1111000 W à + C, CW
                            ELSE 111 REG ! SW
                            THEN INIT ;
 -->
```

BLOC # 19

```
( Assembleur FORTH pour 8086  -arithm(tique-    page 9/13 )

: MUL 11110110  W à + C, 100 REG ! BYTE2 ;

: IMUL 11110110 W à + C, 101 REG ! BYTE2 ;

: DIV 11110110  W à + C, 110 REG ! BYTE2 ;

: IDIV 11110110 W à + C, 111 REG ! BYTE2 ;


: NOT 11110110 W à + C, 10   REG ! BYTE2 ;

: STC 11111001 C, ;       : CLC 11111000 C, ;

-->
```

BLOC # 20

```
( Assembleur FORTH pour 8086   - logique -          page 10/13)

: AND 100000 D à + W à + C, BYTE2 ;
: ANI REG à 0= MOD à 11 = * IF 100100 W à + C, CW
                            ELSE 100 REG ! SW
                            THEN INIT ;

: OR  1000 D à + W à + C, BYTE2 ;
: ORI REG à 0= MOD à 11 = * IF 1100 W à + C, CW
                            ELSE   1 REG ! SW
                            THEN INIT ;

: XOR 110000 D à + W à + C, BYTE2 ;
: XRI REG à 0= MOD à 11 = * IF 110100 W à + C, CW
                            ELSE 110 REG ! SW
                            THEN INIT ;  -->
```

BLOC # 21

```
( Assembleur FORTH pour 8086 - opération chaines- page 11/13 )

: REP 11110010 C, ;                 : REPE 11110011 C, ;

: MOVS 10100100 W à + C, INIT ;   : CMPS 10100110 W à + C, INIT ;

: SCAS 10101110 W à + C, INIT ;   : LODS 10101100 W à + C, INIT ;

: STOS 10101010 W à + C, INIT ;

: CLD  11111100 C, ;               : STD  11111101 C, ;

: LOOP 11100010 C, C, ;            : JCXZ   11100011 C, C, ;

: LOOPZ 11100001 C, C, ;           : LOOPNZ 11100000 C, C, ;
-->
```

BLOC # 22

```
( Assembleur FORTH pour 8086-Transfert inconditionel-page 12/13)

: INT 11001101 C, C, ;        : INT3 11001100 C, ;

: CALL R/M à 1000 = IF 11101000 C, ,
                    ELSE 11111111 C, 10 REG ! BYTE2 THEN ;

: CALLF R/M à 1000 = IF 10011010 C, , ,
                     ELSE 11111111 C, 11 REG ! BYTE2 THEN ;

: JMP  R/M à 1000 = IF 11101001 C, ,
                    ELSE 11111111 C, 100 REG ! BYTE2 THEN ;

: JMPF R/M à 1000 = IF 11101010 C, , ,
                    ELSE 11111111 C, 101 REG ! BYTE2 THEN ;
: JMPS 11101011 C, C, ;                          -->
```

```
BLOC # 23
( Assembleur FORTH pour 8086  -sauts conditionels-  page 13/13 )

: JZ 1110100 C, C, ; : JL 1111100 C, C, ; : JLE 1111110 C, C, ;
: JB 1110010 C, C, ; : JBE 1110110 C, C, ; : JPE 1111010 C, C, ;
 : JO 1110000 C, C, ; : JS 1111000 C, C, ; : JNZ 1110101 C, C, ;
 : JGE 1111101 C, C, ; : JG 1111111 C, C, ; : JAE 1110011 C, C,
; : JA 1110111 C, C, ; : JPO 1111011 C, C, ; : JNO 1110001 C, C,
 ; : JNS 1111001 C, C, ;

: INTO 11001110 C, ;    : IRET 11001111 C, ;
: STI  11111011 C, ;    : CLI  11111010 C, ;

: NOP  10010000 C, ;

FORTH DEFINITIONS      DECIMAL
( Assembleur version 1.00 écrit le 4 Août 1984 / F.LAFAIX     )
```

FORTH Conversion AN/NA

par Roland JEANNIN

Ce programme fonctionne a l'aide des cartes entrées/sorties de la marque Jagot & Leon (E 101 & E 103) .Il permet de saisir des échantillons analogiques à une vitesse comprise entre 10 et 12000 échantillons par seconde,et de les stocker à raison de 20000 éch. en même temps ce signal peut être reconverti en subissant un décalage programmable . On peut donc obtenir des effets d'écho ou de déphasage (de 1 microseconde a 1 seconde). Plusieurs fonctions d'affichage sont possibles par exemple l'affichage de 20 lignes analogiques totalisant 10000 échantillons. Chacune de ces lignes peut être affichée individuellement. Puis on peut aussi n'afficher qu'un éch. sur deux ou moins.

Mots permettant de mémoriser:

- OP1 : Paramêtres : Nombre de fois qu'on pratique le processus entier (pour travailler en continu en audio fréquences avec déphasage) de 1 a 30000
  Temporisation pour diminuer la fréquence d'échantillonnage ex:3 correspond à 12000 éch./sec
  Retard pour ré-injecter le signal pour un écho ou un déphasage.
- OP3 : Parm.:Durée de processus
  Décalage en microsecondes (MOS) ou millisecondes (MS) du retard.
  Fréquence d'échantillonnage en éch./sec
- OP4 : Charge en mémoire les parm. d'OP3.
- DECAL : complète l'opération OP3+OP4.

OP3+OP4+DECAL forme ume opération equivalente à op1 avec des parm. differents .

Mots permettant de tracer:

- TRACER : Trace 20 lignes analogiques (Aucun parm.)
- TR1 : parm.: Ieme ligne analogique
  Dilatation (n'affiche que 1/n éch.)
- TR2: cas particulier de TR1.affiche la premiere ligne
- TR3: ''   ''   ''

Mots servant à garder sur disque :

- >DISC : sauve 20000 éch. aux écrans-disc 160 a 180
- DISC> : redonne les éch. depuis la disquette

EXEMPLES

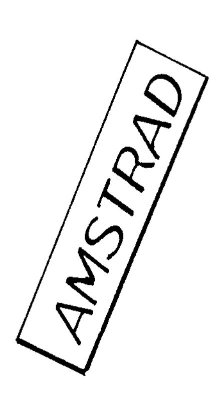
2 5 10000 OP1 :Repasse 2 fois la mem.avec une tempo qui permet une fréquence d'environ 11000 éch./sec.Le retard à la sortie du convertisseur N/A sera d'environ .6 sec.

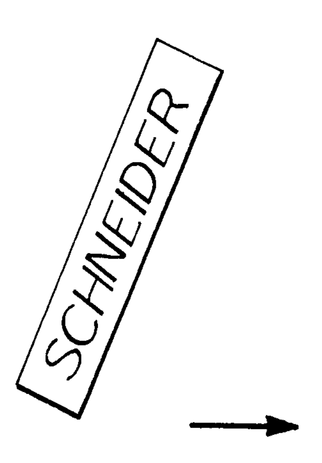
10 30 MS 5000 OP3 OP4 DECAL : opération qui dure 10 sec. de retard à la reconversion =30 millisec. avec une fréquence d'éch de 5000 par sec.

10 2 TR1 : affiche les lignes 10 et 11 à raison de 1 éch sur 2

L'écran 45 est un petit assembleur complet sans les mnémoniques mais avec des labels et des jumps absolus et relatifs.

Renseignements : Jeannin tel 74 27 02 09 Isle d'Abeau





```
SCR # 45
( Langage Machine : calculs auto adresses + utilitaires )
: VAR VARIABLE ;   : CT CONSTANT ; 0 VAR LMS0 0 VAR IDL 30 ALLOT
0 VAR IR 200 ALLOT  IR VAR PI  0 VAR CTI  : DEC DECIMAL ;
: >I PI @ ! 2 PI +! ;   : I> -2 PI +! PI @ @ ;   : IL 2 * IDL + ;
: LMD -2 ALLOT LATEST PFA DUP 2 - ! SP@ LMS0 ! HEX ;
HEX  0 VAR SPI LMD ED C, 7B C, LMS0 , C3 C, 12C ,
: LMC SP@ 2 - LMS0 @ 2 - 2DUP < IF DO I @ DUP ABS FF > IF , ELSE
 C, THEN -2 +LOOP THEN SPI ;
: LMF DEC C3 12C LMC CTI @ 1+ 0 DO I> DUP FF > IF
 100 - IL @ I> 2 - ! ELSE IL @ I> 1 - DUP ROT SWAP - 1 - SWAP
 C! THEN LOOP 0 CTI ! IR PI ! ;
: ** >R LMC HERE R> IL ! ;
: JR DUP >R LMC R> HERE >I >I 1 CTI +! ;
: JA 100 + JR ;  DEC

;S

SCR # 46
( Analogique/digital page 1/4 )
0 VARIABLE REG 19998 ALLOT  1 VARIABLE FOIS
1 VARIABLE TEMPO  REG VARIABLE REGD  0 VARIABLE ACQ
REG 20000 + VARIABLE REGF
HEX F9E0 CONSTANT ANA  F9F1 CONSTANT DIG DECIMAL
;S


SCR # 47
( Analogique/digital page 2/4 )
0 VARIABLE DECAL LMD C5 F3
0 ** 21 REG ED 5B REGD
1 ** 3A ACQ FE 00 20 2 JR 01 ANA ED
      78 77 AF ED 79 18 4 JR
2 ** 3E 04
3 ** 3D 20 3 JR
4 ** 01 DIG 1A ED 79 AF ED 4B TEMPO
5 ** 0B 78 B1 20 5 JR 3A REGF
      BB 20 6 JR 3A REGF 1+ BA 20 6 JR
      11 REG 1 -
6 ** 13 3A REGF 23 BD 20 1 JR 3A REGF 1+
      BC 20 1 JR ED 4B FOIS 0B ED 43
      FOIS 78 B1 20 0 JR FB
C1         LMF
;S

SCR # 48
( Analogique/digital page 3/4 )
: OP1 ( Fois tempo retard--) REGF @ SWAP - REGD ! TEMPO ! FOIS !
  DECAL 7 EMIT ;
: SAUVE 500 CMOVE UPDATE FLUSH ;
: >DISQ 40 0 DO REG I 500 * + I 320 + BLOCK SAUVE LOOP ;
: DISQ> 40 0 DO I 320 + BLOCK REG I 500 * + SAUVE LOOP ;
0 VARIABLE WIN  0 VARIABLE DIL
: TRACER CLS 20 0 DO 19 I - WIN ! 500 0 DO REG I + 19 WIN @ -
  500 * + C@ 10 128 */ WIN @ 20 * + I PLOT LOOP LOOP
  1 75 LOCATE 7 EMIT KEY DROP ;
1000 CONSTANT MS  1 CONSTANT MOS
: TR1 ( Ieme DIL--) DIL ! CLS DUP DUP 640 DIL @ / + SWAP
  DO REG I + C@ 400 256 */ OVER I SWAP - DIL @ * PLOT LOOP ;
: TR2 0 1 TR1 ;   : TR3 0 10 TR1 ; ;S



SCR # 49
( Analogique/digital page 4/4 )
: OP3 ( duree, decalage, (mos ou ms), freq d'ech-- )
  >R 14388 100 R> */ 96 - 10 / 1 MAX DUP TEMPO !
  7 * 63 + DUP >R */ REG SWAP - 20000 + REG MAX REGF MIN REGD !
  50 R> */ 1 MAX FOIS ! ;
: OP4 TEMPO @ . REGD @ . FOIS @ . ;
;S

```

ALGORITHME

Les fractions continues.
Approximation d'un nombre par une fraction.

Quelques définitions, pour commencer:

nombre rationnel: quotient de deux entiers.
nombre algébrique: racine d'un polynôme à coefficients entiers.
nombre réel: nombre mathématiquement défini.

De plus, dans cet article, la précision, positive, sera le quotient de la valeur divisée par l'écart: différence entre la valeur exacte et approchée.

Pour différentes raisons, il a été préféré, en FORTH, de représenter un nombre par un quotient d'entiers, plutôt que par la représentation en virgule flottante, ce qui augmente la précision pour les nombres moyens, mais la diminue pour les petits ou grands nombres.

Comment touver la fraction qui approxime le mieux un nombre donné à l'avance?

DEVELOPPEMENT EN FRACTION CONTINUE

La clef du problème est le développement en fraction continue: à partir du nombre donné à l'avance, u_0, construisons la suite:

$$u_0 = a_0 + 1 / u_1$$
$$\ldots\ldots\ldots$$
$$u_n = a_n + 1 / u_{n-1}$$

a_i est un entier, obtenu par troncature, cas qui ne sera pas examiné ici, ou par arrondi: c'est l'entier le plus proche de u_i, donc avec une précision de $2 * a_i$; la prise en compte de a_i dans la détermination de u_0 multiplie la précision par au moins $2*a_i$, et le majorant du numérateur sera multiplié par a_i.

Pour un nombre rationnel, la suite est finie: le dernier nomre est un entier, pour un nombre algébrque, la suite des a_i est périodique.

Pour savoir à quelle profondeur s'arrêter, il suffit de comparer le majorant au plus grand nombre codable en 15, 16, 31 ou 32 bits.

Il suffit alors, pour remonter les calculs, de construire:

$$v_{n+1} = 1$$
$$v_n = u_n$$
$$v_{n-1} = u_{n-1} * v_n + v_{n+1}$$
$$\ldots\ldots\ldots\ldots\ldots$$

$$u_0 = v_0 / v_1$$

Cet algorithme est applicable directement pour cadrer sur 15 ou 16 bits un quotient d'entiers de 31 ou 32 bits; c'est une application de la méthode générale d'évaluation d'une grandeur inconnue, par rapport à une unité, uniquement au compas: la grandeur u_0 est évaluée par rapport à l'unité u_1, reste un écart u_2, inférieur à la moitié de u_1, puis c'est l'unité qui est évaluée par rapport à l'écart u_2:

$$u_0 = n_0 * u_1 + u_2$$
$$u_1 = n_1 * u_2 + u_3$$
$$\ldots\ldots\ldots\ldots\ldots$$

La convergence est plus rapide que la dichotomie. Cette méthode peut avoir d'autres applications, comme le calcul de logarithme.

EXEMPLE DE CALCUL

Passons à la pratique. Il n'a pas été nécessaire d'écrire un programme, d'usage peu fréquent, néanmoins, avis aux amateurs: il pourra intéresser plus d'un. Les premiers calculs devront être précis: les calculs à 12 décimales sont possibles avec une calculette ordinaire à 8 chiffres, voici comment: par exemple, pour inverser u = 0.2817 1817 155 , un premier calcul donne: 1/u = 3.5496 467 .
Calculons:

3549 *1817.155 / 10000 M+ Décimales: 0.9083
3549 * 2817 M+ Résultat: 999 8177.9
10000 -8177 - 0.9083 * 10000 / 2817.1817
= 6467.7819

d'où 1 / u = 3.5496 4677 819 .

Je propose comme exercice de développer en fraction continue les nombres suivants: racine carrée de 2, 5, 13, et le nombre:3.3781 9799 7775 3058 9543 9377 0856 5072 3025 0840

L'approximation de e s'obtient par les calculs suivants:

i	u_i	a	m	p
0	u_0=2.7182 8182 845	3	3	6
1	u_1=1/(3-2.7182 ..) =-3.5496 4677 82	-4	12	48
2	1/(-4+3.5496 4677) = 2.22047928 51	2	24	192
3	4.5355 7348 8	5	120	1920
4	-2.1531 9318	-2	240	7680
5	-6.5277 056	-7	1680	9 2160
6	2.1173 24	2	3360	36 8640
7	8.5231 4	9	3 0240	663 5520
8	-2.0984	-2	6 0480	2654 2080
9	-10.166	-10	60 4800	5 3084 1600
10	-6.018	-6	362 8800	$6.37\ 10^9$
11	-56.4	-56	2 0321 2800	$713.45\ 10^9$
i	u_i	a_i	majorant	précision

→

Pour les déterminations d7 et d8 de u_0, s'arrêtant à la 7 et 8ème ligne, les calculs sont:

4	5	6	7	8	9	i
-2	-7	2	9	-2	-9	a_i
267	-124	19	9	1		v_i^7
-506	235	-36	-17	-2	1	v_i^8

Précision	0	1	2	3	i
	3	-4	2	5	a_i
0.49 10^9	-25946	-9545	2689	1211	v_i^7
9.5 10^9	49171	18089	-5096	-2295	v_i^8

La précision correspond à 29 bits pour d7 et 34 pour d8.

Rien de mystérieux, comme on peut le voir, sur la provenance de fractions telles que 22 / 7 ou 355 / 113 ...

FORTH DEFER en 79-Standard

par Marc PETREMANN

```
: ?INIT ( ---)
CR ." Vecteur non initialisé " 3 ERROR ;

: DEFER
CREATE [ FIND ?INIT ] LITERAL ,
DOES> @ EXECUTE ;

: ['] ( ---)
FIND [COMPILE] LITERAL ; IMMEDIATE

: IS    ( cfa ---)
FIND 2+ STATE @
IF   [COMPILE] LITERAL COMPILE !
ELSE !             THEN ; IMMEDIATE

;S Création de mots vectorisés au for-
mat 83-STANDARD. Exemple:

DEFER TEST        crée un en-tête TEST
['] CLS IS TEST   affecte le CFA de CLS
                  au PFA de TEST. Doréna-
                  vant, TEST exécute CLS
```

Thomson TO7-MO5

Dans le prochain numéro de JEDI:

Un système expert complet avec moteur d'inférence d'ordre 1 écrit en langage FORTH

Ce programme appelé FORTHLOG est extensible et traite les variables numériques et alphanumériques

Il est implantable sur tous les systèmes (implantation d'origine sur COMMODORE 64) F.I.G., 79-STANDARD et 83-STANDARD.

UNE GESTION PAR BLOCS

Ce programme est inspiré d'un article paru dans le mensuel Dr.DOBB'S Journal paru sous le titre "FILE MAINTENANCE IN FORTH". Il est réécrit ici sous une forme compatible avec les commandes du système THOMSON TO7, TO7-70, et peut être adapté sans inconvénients sur la plupart des systèmes FORTH au standard FIG ou 79-STANDARD.

Je ne vais pas pour une fois expliquer le fonctionnement des différentes commandes de ce programme, mais brosser un tableau général des notions mises en jeu.

L'enregistrement dans les blocs: l'éditeur du langage FORTH agit sur le contenu des blocs. Ces blocs correspondent physiquement à des portions du support d'enregistrement de la disquette (ou cassette dans le pire des cas).

Par convention, un bloc correspond à un espace mémoire de 1024 octets, soit 1 Koctets. La position en mémoire de ce bloc peut être relevée en tapant la commande n BLOCK . où n correspond au numéro du bloc auquel on désire accéder.

En général, le numéro de bloc et le numéro d'écran est le même, mais ce n'est pas un cas général. Ainsi, sur THOMSON, la position en mémoire du contenu de l'écran 15, pour exemple, peut être mis en évidence en tapant la commande 15 BLOCK 1024 DUMP . Dans le cas de l'AMSTRAD, il faudra taper 15 B/SCR * BLOCK 512 DUMP. Le mot B/SCR détermine le nombre de blocs par écrans, le mot B/BUF délivre le nombre d'octets par BLOCK. La relation n B/SCR * BLOCK B/BUF DUMP sera la plus générale et s'applique donc au THOMSON comme à l'AMSTRAD (ou les autres systèmes).

Pour enregistrer une donnée dans un bloc, et et par conséquence sur le disque (ou cassette), on peut faire une commande du genre:

```
donnée position n BLOCK + C! ( ou ! ) UPDATE
```

et SAVE-BUFFERS ou FLUSH (selon systèmes). Pour récupérer notre donnée, on peut taper

```
position n BLOCK + C@ ( ou @ ) .
```

L'intérêt de cette méthode n'est pas évident au premier abord. Pourtant elle reste la plus pratique pour le stockage de données à accès direct y compris pour les données sur cassette, car on peut faire transiter plusieurs blocs simultanément en mémoire (jusqu'à 20 unités sur THOMSON), ce qui permet de gérer les blocs à la manière des "RAMS DISK". Mais voyons plus en détail l'implantation de données tel qu'on l'entend quand on parle de fichiers.

Rappel sur la structure des fichiers
Un fichier est un ensemble ordonné de données dont l'accès est réalisé à l'aide de clés, par accès séquentiel ou par accès direct. Dans le cas nous intéressant, on étudiera l'accès direct, étant entendu qu'il vous est loisible de modifier le programme listé plus loin pour réaliser un accès séquentiel ou par une table d'index. Ces données sont regroupées par enregistrement; c'est le premier sous-ensemble du fichier. Chaque enregistrement est divisé en champs. En BASIC, c'est l'instruction FIELD qui permet l'affectation de la taille et du type de chaque champ.

Voyons comment, en langage FORTH, nous allons implanter les données de notre fichier en tenant compte de ce qui est exprimé ci-dessus.

Implantation d'un enregistrement dans un bloc
Prenons comme bloc de départ de notre fichier le bloc bl.orig. et dim.enr la taille d'un enregistrement. Le nombre d'enregistrements maximum contenus dans un bloc est donné par la relation

```
B/BUF dim.enr /
```

Soit RLEN la variable contenant la taille d'un enregistrement (RECORD LENGTH), et REC/BLK le nombre maximum d'enregistrements, on peut initialiser REC/BLK par:

```
B/BUF RLEN @ / REC/BLK !
```

Dans le cas du THOMSON, B/BUF peut être remplacé par le litéral 1000 et par 1024 pour les systèmes autre qu'AMSTRAD (512 dans son cas).

On localisera le bloc contenant l'enreristrement n en exécutant:

```
n REC/BLK @ /MOD
```

où le qotient représente le numéro de bloc contenant l'enregistrement recherché et le reste correspondant à la position de l'enregistrement dans ce bloc. Soit RNO le numéro d'enregistrement, BBLOCK le numéro du premier bloc du fichier, l'adresse de l'enregistrement recherché sera obtenue par:

```
n REC/BLK @ /MOD  ( n --- qot res )
BBLOCK @ + BLOCK SWAP  ( qot res --- res adr)
RLEN @ * +  ( res adr --- adresse d'enreg.)
```

Le résultat est stocké dans RLOC (RECORD LOCATION).

Les enregistrements ne peuvent chevaucher deux blocs, car la fonction de division en FORTH ne délivre que le qotient plancher (20 10 / donne 2, mais 29 10 / donne aussi 2)

Donc, dans le meilleur des cas, la place perdue d'un bloc au suivant sera nulle, sinon elle sera de RLEN-1 dans le pire des cas. On optimisera donc en prenant comme taille d'enregistrement une valeur égale à B/BUFF divisé par REC/BLK.

Mais cette optimisation n'est pas obligatoire.

Gestion des champs dans un enregistrement
Le surnom du programme est fBASE-I, ce qui a un léger rapport à dBASE II, mais n'a pas la prétention d'égaler ce progiciel. Cependant, fBASE I lui est supérieur sur un point, celui de la présentation des données. En effet, en dBASE-II la saisie des données est réalisée en "plein écran", mais avec pour seule identification des champs, un libellé de taille réduite et souvent symbolique. Si on désire améliorer cet état de chose, il faut créer un fichier de commandes. Pour fBASE-I, la saisie des données est également réalisée en "plein écran", mais avec la possiblilité de positionner séparément les champs de saisie et d'identification. Exemples: on peut mettre le champ de saisie du nom à la la suite de l'identification:

NOM:______________

ou bien deux lignes en dessous:

NOM

____

et ainsi de suite pour l'ensemble des champs de saisie et leurs identificateurs.

Dans dBASE-II, l'identificateur d'un champ correspond au nom qui lui a été attribué lors de la création de la structure du fichier (commande CREATE et LIST STRUCTURE). L'identificateur ne peut contenir ni espace, ni signes autres que les caractères aplhanumériques. Avec fBASE-I, l'identificateur peut être constitué avec l'ensemble des caractères alphanumériques disponibles au clavier, excepté le séparateur ".

Un exemple de gestion des champs de l'enregistrement est donné dans le bloc 23. Considérons la ligne suivante:

```
00 00 00 07 32 9 00 NOM"
```

On indique au système FORTH que le champ identifié par NOM est paramétré par:

00 00 position en yx de l'affichage de l'identificateur. →

Ø7 position en yx du champ de saisie à l'écran.

32 taille du champ de saisie

Ø9 type des données à saisir (ici alphanumérques).

ØØ position dans l'enregistrement. Cette valeur est égale à la somme de la taille du précédent champ et de sa position. Dans le bloc 23, à la seconde ligne de la structure du fichier, le champ PRENOM" débute au 32 octet de l'enregistrement (valeur de la taille du champ NOM"), et fait 22 octets de longueur.

NOM" correspond à l'identificateur du premier champ de l'enregistrement. Il peut être constitué de 255 caractères maximum, espaces compris et se termine toujours par le caractère " . On aurait aussi bien pû définir l'identificateur

PATRONYME DE L'ADHERENT"
ce qui est impossible avec dBASE-II.

Voici ce que donne à l'écran une page de saisie de fBASE-I:

```
NOM ______________________________
PRENOM ____________________  SEXE ___
ADRESSE
__________________________________
COMPL.ADRESSE
__________________________________
CDE.POST _____
COMMUNE
__________________________________
MOIS __ ANNEE ADHESION ___
```

<u>Structure d'un enregistrement</u>

L'exemple fourni par le bloc 23 montre clairement la structure d'un enregistrement. Chaque ligne correspond aux paramètres d'un champ. Il y a 9 champs. On définit donc la structure du fichier par 9 PBUILD ADHER

PBUILD est un mot de définition de structure. Il doit être suivi par le nom de la structure, ici ADHER, et le détail de chaque champ.

Le fichier proprement dit est paramètré par le mot qui suit, ici GESTION. Ce mot est défini par l'utilisateur. Les paramètres qui précèdent IREC (pour INIT-RECORD) signifient que l'on initialise un fichier débutant au bloc 200 et dont les enregistrement ont 186 octets de longueur:

```
200 186 IREC
```

Puis divers paramètres sont transférés depuis le début de l'enregistrement Ø vers PHIGH. En fait, la fiche Ø doit être initialisée "à la main" par:

```
Ø Ø RECORD ! UPDATE SAVE-BUFFERS
```

en ayant pris la précaution d'initialiser la position du fichier par 2ØØ BBLOCK !

Le détail de la structure d'un enregistrement n'est pas automatisé. La position des différents champs reste à votre charge. Il est donc nécessaire de connaître les différents types de données gérées par fBASE-I:

type 9: données alphanumériques; tout caractère alphanumérque, à concurence de la taille du champ de saisie.

type 1: donnée numérique sur 1 octet. Sera affichée dans un champ justifié à droite.

type 2: donnée numérique 16 bits. Sera affichée dans un champ justifié à droite.

type 3: donnée numérique 16 bits; affichage de type "financier" avec virgule décimale.

type 4: donnée numérique 32 bits; sera affichée dans un champ justifié à droite.

type 5: donnée numérique 32 bits; affichage identique type 3.

La taille minimale de l'enregistrement pour chaque type de données est:

type 1: taille du champ de saisie.
type 2: 2 octets.

type 3: 2 octets.

type 4 et 5: 4 octets.

<u>Saisie du listing:</u>

La saisie du listing ne doit pas poser de problème particulier. Les mots spécifiques THOMSON sont les suivants:

ASC dépose sur la pile le code du premier caractère de la chaine qui précède. Exemple:

```
" A" ASC =
```

équivaut à

```
65 =
```

CREATE équivalent 79-STANDARD de <BUILDS.

INVCOLOR bascule l'affichage en "vidéo inverse". Peut être omis à la frappe.

LOCATE positionnement en ligne y, colonne x. Equivaut à LOCATE sur AMSTRAD. Pour les autres systèmes, voir si implanté, sinon il faut créer ce mot en se servant d'une séquence US (code ACSII US , càd 31 EMIT)

<u>Menu de présentation:</u>

```
        MAINTENANCE DE FICHIER

1 - Ajouter
2 - Lister
3 - Afficher fiche
4 - Changer fiche
5 - Quitter
```

<u>Performances actuelles du programme fBASE-I</u>

Après essai sur THOMSON TO7-70, fBASE-I effectue une recherche et un affichage du contenu contenu d'un enregistrement plus rapidement que dBASE-II. Par contre, il ne permet pas l'édition et la recherche par critères, ce qui est normal vu l'état d'avancement du programme actuel. fBASE-I occupe moins de 3,5 K. Un même fichier peut être édité avec différentes structures (présentations), soit partiellement, soit totalement.

Il fonctionne également en version cassette. Dans ce cas, il est conseillé de préparer le système avec un maximum de buffers. Dans le cas du THOMSON, on tapera:

```
20 BUFFERS
: CLOAD  1+ SWAP DO I BLOCK DROP LOOP ;
BBLOCK DUP 20+ CLOAD
```

ce qui charge 20 blocs d'un coup. L'amplitude d'accès aux différents enregistrements sera alors limitée, mais en considérant l'espace mémoire des blocs comme un disque virtuel, on bénéficie quand même de l'accès direct dans cet espace. Il faudra également modifier la sauvegarde sous la forme:

```
: CSAVE 1+ SWAP DO I BLOCK UPDATE
    DROP SAVE-BUFFERS LOOP ;
```

<u>Adaptations possibles</u>

En changeant les caractéristiques de taille de bloc, on peut adapter les accès disque à la taille des blocs physiques de la disquette; ceci est particulièrement vrai pour le FORTH AMSTRAD où les numéros de blocs disponibles vont de 2 à 180 pour le drive Ø.

En rajoutant des routines de sélection, recherche, tri et indexation, vous réinventerez dBASE II.

Suite texte à la fin du listing.



```
SCR: 10
( fBASE-I   écran 1                    MP10jun85)

VARIABLE RLOC     0 RLOC      !
VARIABLE RLEN     0 RLEN      !
VARIABLE RNO      0 RNO       !
VARIABLE REC/BLK  0 REC/BLK   !
VARIABLE BBLOCK   0 BBLOCK    !

: IREC ( BBLOCK RLEN --- )
 RLEN !   BBLOCK ! ;

: RECORD ( RNO --- ADR )
 RNO !   1024 RLEN @ /   REC/BLK !
 RNO @ REC/BLK @ /MOD
 BBLOCK @ + BLOCK SWAP RLEN @ * +
 DUP RLOC ! ;

VARIABLE DFLAG

: ?DIGIT ( N --- FL )
 DUP 47 > SWAP 60 < AND ;

SCR: 11
( fBASE-I   écran 2                    MP10jun85)

: ?NUMBER   ( LOC --- FL )
0 DFLAG !   DUP 11 + SWAP

DO
  I C@ ?DIGIT
  IF
    1 DFLAG !
  ELSE
    I C@ 32 = I C@ 0= OR
    IF
      LEAVE
    ELSE
      I C@ 46 = 0=
      IF
        0 DFLAG ! LEAVE
      THEN
    THEN
  THEN
LOOP
DFLAG @ ;

SCR: 12
( fBASE-I   écran 3                    MP10jun85)

: ASK ."  (O/N) " KEY " O" ASC =  ;

: LEN ( ADR --- COUNT )
 255 0
  DO
   DUP I + C@ 0=
   IF
     I LEAVE
   THEN
 LOOP
 SWAP DROP ;

: WAIT ( --- )    CR CR
." Appuyer une touche pour continuer"
KEY DROP ;
```

```
SCR: 13
( fBASE-I   écran 4                    MP10jun85)

VARIABLE PHIGH      0 PHIGH    !
VARIABLE FNO        0 FNO      !
VARIABLE FLOC       0 FLOC     !
VARIABLE PDONE      0 PDONE    !
VARIABLE PLOC       0 PLOC     !
VARIABLE PMAX       0 PMAX     !

: PINIT ( TBLLOC --- )
 DUP PLOC !
 C@ PMAX C!
 0 PDONE !   0 FNO ! ;

SCR: 14
( fBASE-I   écran 5                    MP10jun85)

: PBUILD ( NO-OF-FLDS --- )
CREATE  DUP C, 0 DO
32 WORD NUMBER DROP C,
        ( ligne d'affichage des données)
32 WORD NUMBER DROP C,
        ( colon.d'affichage des données)
32 WORD NUMBER DROP C,
        ( ligne de saisie              )
32 WORD NUMBER DROP C,
        ( colonne de saisie            )
32 WORD NUMBER DROP C,
        ( longueur de données          )
32 WORD NUMBER DROP C,
        ( type des données             )
32 WORD NUMBER DROP C,
        ( position des données ds fiche)
34 WORD C@ 1+ ALLOT
        ( texte à afficher             )
LOOP DOES> ;

SCR: 15
( fBASE-I   écran 6                    MP10jun85)

: FFIND ( FNO --- FLOC)
 PLOC @ 1+ FLOC ! ?DUP
IF
  0
  DO
    FLOC @ 7 + DUP C@ + 1+ FLOC !
  LOOP
THEN
FLOC @ ;

: FDISPLAY ( FLOC --- )
 DUP DUP C@ SWAP 1+ C@   SWAP
 LOCATE 7 + COUNT
 INVCOLOR TYPE INVCOLOR
 FLOC @ 2+ C@ FLOC @ 3 + C@    SWAP
 LOCATE FLOC @ 4 + C@ 0
 DO  95 EMIT LOOP ;

: PDISPLAY ( --- )
 CLS PMAX C@ 0
 DO I FFIND FDISPLAY LOOP ;
```

→

```
SCR: 16
( fBASE-I  écran 7                    MP10jun85)

: ?FDATA ( FLOC --- FLOC TRUE ou )
         ( FLOC --- FLOC FALSE    )
DUP 4 + C@ 0= IF  DROP 0  ELSE 1  THEN ;

: FDATA ( FNO --- LOC LENGTH TYPE ou)
        ( FNO --- 0  si pas donnée   )
 FFIND ?FDATA
 IF
   DUP 2+ C@ SWAP 3 + C@ SWAP LOCATE
   RLOC @ FLOC @ 6 + C@ +
   FLOC @ 4 + C@
   FLOC @ 5 + C@
 ELSE
   0
 THEN ;

SCR: 17
( fBASE-I  écran 8                    MP10jun85)

VARIABLE FTYPE   0 FTYPE !

: FTEXT ( LOC OLEN PLEN ---)
        ( Saisie data texte)
 ROT ROT DDUP BL FILL DROP
 PAD SWAP ROT CMOVE ;

: FNUM
 DDROP PAD DUP 11 + SWAP
 DO
   I C@ 0=
   IF
     32 I C!
   THEN
 LOOP
 PAD ?NUMBER
 IF
   PAD 1 - NUMBER
 ELSE
   0 0
 THEN ;

SCR: 18
( fBASE-I  écran 9                    MP10jun85)

: FGET ( FNO --- )
 FDATA ?DUP
 IF
   FTYPE ! PAD OVER EXPECT PAD LEN ?DUP
   IF FTYPE @
     CASE
       9 OF FTEXT                     ENDOF
       1 OF FNUM DROP SWAP C!         ENDOF
       2 OF FNUM DROP SWAP !          ENDOF
       3 OF FNUM DROP SWAP !          ENDOF
       4 OF FNUM ROT D!               ENDOF
       5 OF FNUM ROT D!               ENDOF
     ENDCASE
   ELSE
     DROP DROP
   THEN
 THEN ;

: .Fr
 SWAP OVER DABS
 <# # # 46 HOLD #S SIGN #> TYPE SPACE ;

SCR: 19
( fBASE-I  écran 10                   MP10jun85)

: FPUT ( FNO --- )
FDATA ?DUP
IF
  DUP FTYPE !
  CASE
    9 OF TYPE                ENDOF
    1 OF SWAP C@ SWAP .R     ENDOF
    2 OF SWAP @ SWAP .R      ENDOF
    3 OF DROP @ 0 .Fr        ENDOF
    4 OF SWAP D@ SWAP D.R    ENDOF
    5 OF DROP        D@ .Fr  ENDOF
  ENDCASE
THEN ;

: PGET ( --- )
UPDATE PLOC @ C@ 0 DO I FGET LOOP ;

: PPUT ( --- )
PLOC @ C@ 0 DO I FPUT LOOP ;

: PNEXT ( Enr. suivant à ajouter)
 PHIGH @ 1+ DUP PHIGH ! RECORD DROP ;

SCR: 20
( fBASE-I  écran 11                   MP10jun85)

: PADD ( Ajouter ou changer un enr.)
PDISPLAY PGET 0 ;

: PLIST
CLS
PHIGH @ 1+ 1
DO
  I 3 .R SPACE
  I RECORD 20 TYPE CR
  I 23 MOD 0= IF WAIT CLS THEN
LOOP
WAIT 0 ;

: PSELECT ( Prend un enr. pour trt fut.)
CLS CR
." Entrez le numéro d'enregistrement:"
INPUT DROP  DUP PHIGH @ >
IF  ." Pas d'enregistrement " DROP 0
ELSE RECORD DROP 1
THEN ;

SCR: 21
( fBASE-I  écran 12                   MP10jun85)

: PEND  ( Termine cette tâche)
CR ." Etes-vous sûr " ASK
IF  PHIGH @ 0 RECORD ! UPDATE
CLS ." Sauvegarde des fichiers" CR
SAVE-BUFFERS ." Au revoir" CR 1
ELSE 0
THEN ;
```

→

```
SCR: 22
( fBASE-I  écran 13                    MP10jun85)

: PMENU ( --- )
CLS 10 6 LOCATE
." MAINTENANCE DE FICHIER"
1 10 LOCATE ." 1 - Ajouter" CR
1 12 LOCATE ." 2 - Lister" CR
1 14 LOCATE ." 3 - Afficher fiche" CR
1 16 LOCATE ." 4 - Changer fiche" CR
1 18 LOCATE ." 9 - Quitter" CR
10 20 LOCATE
." Choisissez une option" KEY
CASE
  49 OF PNEXT PADD              ENDOF
  50 OF PLIST                   ENDOF
  51 OF PSELECT
     IF PDISPLAY PPUT WAIT 0 THEN
                                ENDOF
  52 OF PSELECT
     IF PDISPLAY PPUT PGET 0 THEN
                                ENDOF
  57 OF PEND                    ENDOF
  0  SWAP
ENDCASE ;
```

```
SCR: 23
( fBASE-I                              MP10jun85)

9 PBUILD ADHER
00 00 00 07 32 9   00 NOM"
02 00 02 07 22 9   32 PRENOM"
02 32 02 37 03 9   54 SEXE"
04 00 05 00 40 9   57 ADRESSE"
07 00 08 00 40 9   97 COMPL.ADRESSE"
10 00 10 10 05 9 137 CDE.POST"
12 00 13 00 40 9 142 COMMUNE"
15 00 15 05 02 1 182 MOIS"
15 08 15 23 02 1 184 ANNEE ADHESION"

: GESTION
 200 186 IREC
 0 RECORD @ PHIGH ! ADHER PINIT
 BEGIN PMENU UNTIL ;
```

En affectant une variable alphanumérique comme tampon pour chaque fichier, il devient possible d'ouvrir plusieurs fichiers simultanément.

En gérant des étiquettes hiérarchisées, vous réinventerez les fichiers de type ISAM. Exemple d'étiquette hiérarchisée: le champ ETAT-CIVIL, sans paramètre, peut regrouper NOM" et PRENOM". Dans ce cas, l'exécution d'une commande du genre ECRIS NOM" affiche le contenu du champ qui suit NOM"; ECRIS ETAT-CIVIL pourrait afficher le contenu des champs NOM" et PRENOM".

Et je m'arrête là, vous laissant réfléchir et proposer vos extensions, idées, suggestions, etc...

Monsieur Jean Grézel
La Bastide
Vallon des Hirondelles
83200 TOULON

Toulon, le 17 février 1986

Objet : language Forth - Calculs sur 64 bits.

Réf. : The complete Forth (Alan Winfield, Sigma Technical Press)

Messieurs,

Certaines applications m'amènent à souhaiter disposer de mots permettant d'effectuer les quatres opérations sur 64 bits ainsi que l'affichage de résultats en 64 bits (4 fois 16 bits sur la pile).

Le principe de la multiplication est abordé dans le livre cité en référence (page 78) :

si a, b forment un nombre double longueur (32 bits)
si c, d forment un nombre double longueur (32 bits).

En utilisant UM* (un1 un2 -- ud), on peut alors effectuer la multiplication suivante:

```
                        a        b
                        c        d      *
                   -------------------------
                        d*b      d*b
              d*a       d*a      0      +
              c*b       c*b      0      +
     c*a      c*a       0        0      +
-------------------------------------------
     p        q         r        s
```

Le résultat de la multiplication peut donc être obtenu sur 64 bits.

Espérant vivement que cette question, que je pense être d'ordre général, vous intéressera et que très prochainement je pourrais trouver les mots suivants de manipulation de nombres signés de 64 bits (4 fois 16) dans JEDI :

```
   Q+      Q-      Q*      Q/      QMOD      Q/MOD      QNEGATE
et         Q.      Q.R     (affichages de nombres sur 64 bits).
```

Je vous prie de croire, Messieurs, à l'assurance de mes sentiments distingués.

Jean Grézel

PS : J'utilise assez couramment (avec FORTH 83) les mots de ma confection suivants:

```
: DN*    (dn1 dn2 -- dn3)
         >R  ROT  2DUP  UM*  ROT   R>   UM*  DROP  >R  >R
                             -ROT       UM*  DROP  R>  R>   +  + ;

: D/MOD  (dn1 un -- dn2 dn3) \ dn3: quotient a le signe de dn1
         3DUP  D/  2DUP  UM*  4 ROLL    D*   DNEGATE
                   2ROT  D+   2SWAP     ;

sachant que :
: -ROT   (n1 n2 n3 -- n3 n1 n2)
         ROT  ROT ;
: 3DUP   (n1 n2 n3 -- n1 n2 n3 n1 n2 n3)
         2DUP  4 PICK  -ROT ;
et que les mots suivants existent dans mon forth 83 et sont de la forme :
   UM*   (un1 un2 -- ud)
   D*    (dn  un  -- dn)
   D/    (dn  un  -- dn)
```

F. COMPILER, LINKER, EXECUTER.

Vous possédez sur d 1 = "COBOL" voir "D" dans cette étude et sur "d 2 :Link," ce qu'il faut dans un premier temps pour faire tourner un programme.
Nous nous exercons sur SQUARO.COB un programme source que Microsoft met à votre disposition sur le Master-Cobol - Volume 2. (Il extrait la racine carrée du nombre que vous lui donnez et cela par la méthode de Newton, celle que vous employez vous même. Je prends le plus grand carré etc...)_
Nous nous exercerons ensuite sur notre programme source personnel.

1) COMPILER

a) d'abord : Charger SQUARO.COB sur "Link" si ce n'est déjà fait par votre étude antérieure d 2 Link en B : Master Cobol 2 en A.

```
A> B: [R] et vous êtes en B:→B>
B> PIP B: .= A: SQUARO.COB [R]
B> DIR     pour vérifier.
```

b) COMPILER : "d2 Link" en "B:" ; "d 1 COBOL" en "A:". En tapant simplement:

```
B> A: COBOL [R]  ou si vous êtes en A
A>    COBOL [R]
```

Mais décidément j'aime mieux que vous soyez en B. Vous obtinez * ☐ curseur.
et allez pouvoir parfaire l'ordre.
Les 5 programmes nécessaires (COBOL.COM, COBOL 1.OVR, COB..., COBOL 4.OVR) vont s'éxécuter à la file, et en B vous obtiendrez en plus du SQUARO.COB :
SQUARO.PRN programme source avec les annotations d'erreurs de compilation. (Les petites ligne par ligne, les grosses à la fin).
et SQUARO.REL 1re compilation qui servira de base au linkage (chainage des routines).

Donc l'ordre est

```
*, = B: SQUARO.COB/R/L [R]
```

"," obligatoire, sans cela, compilation à vide sans édition .PRN
B: pour dire que Squaro est en B: et qu'il faut également y placer Squaro.PRN et Squaro.REL.
R pour dire qu'il faut écrire SQUARO.REL.
L pour dire qu'il faut écrire SQUARO PRN.
ET N I NI C'est fini.
Il existe d'autres versions de l'odre, mais c'est celle qui pour moi m'a donné moins de souci. Et ça marche, à chaque coup, même si votre "Source" est très mauvaise.
Il faut environ 2 minutes et moins si le programme est très court.
On vous donne le nombre de fautes :

```
n 1 FAULT (S)      n 2 WARNING(S)
(WARNING = AVERTISSEMENTS)
```

Pour SQUARO, vous aurez :

```
0 FAULT (S)      0 WARNING (S)
```

Pour voir vos fautes faites

```
B> ED  SQUARO.PRN  [R]
→ 1uu*☐
→ 1uu *150 A [   ]
```

Pour avoir le listing en mémoire, voir ED.
Puis par [R] , [R] , [R] .
Vous faites défiler votre programme avec les remarques de compilations. Pour Squaro, il n'y en a pas, mais pour le votre ?? on verra.

Bien sur, vous pouvez éditer ..XXX.PRN sur l'imprimante.

```
B> PIP  LPT: = B: SQUARO.PRN [R]
```

Bien sur, sur votre programme personnel, c'est le moment de rectifier les fautes sur votre

```
XXX.COB par  >ED     XXX.COB
```

Et vous connaissez la suite voir "E" et d'effacer votre XXX.PRN et votre XXX.REL de garder votre XXX.COB (rectifié) et de recommencer une compilation. Voir plus haut - etc, etc...
jusque! - - -

```
0 FAULT          0 WARNINGS.
```

---

2) Linker (chainer les routines).

Il faut repartir d'un "bon" XXX.REL c'est à dire, 0 FAULT... et faire le chainage des routines au moyen de L 80 sur les routines que vous trouvez (dans un premier temps) dans COBLIB et dans CRTDRV.

D'autres routines, ou d'autres sous programmes sont parait-il dans LIB.COM et .... je ne sais lesquelles ..... et peut être ailleurs!

D'ailleurs, je ne sais absolument pas non plus quelles sont les routines en COBLIB !! et il y en a pour 35 K de mémoire! Si Microsoft le sait, lui, peut être pourrait-il nous le dire ?. Par contre il nous apprend en 24 pages ! ce qu'il faudrait y avoir dans CRTDRV (et CRTDRV ne fait que 1 K !) Je vous rassure : CRTDRV marche avec l'écran d'Apple normal (bien que celui-ci n'ait que 40 colonnes et que CRTDRV est programmé pour 80).

Il faudrait peut-être aussi pouvoir aggreger vos routines personnelles. C'est possible en

a) écrivant en Macro assembleur CPM
b) compilant ou plutôt assemblant, par Macro 80 (M 80 sur Master Cobol n° 2) de la même façon que nous l'avons fait avec cobol et ensuite,
c) Linker par L 80.

Vous voyez nous pourrons encore bien nous amuser.

Donc notre linkage ————→ ...

Pour la 1re fois, nous nous exerçons sur SQUARO.REL que nous avons obtenu en 1°/ sur la disquette d 2, sur d 2 nous avons également COBLIB et CRTDRV. Nous mettons en A et

a) A> L 80 [R] return
   *□

b) chainage (le chainage sur COBLIB est très long. Soyez très patient).

* A: SQUARO.REL, A: COBLIB/S, A: CRTDRV/S [R]
* □→curseur (après un assez longtemps)
Restez en * et

c) obtention de SQUARO.COM le programme objet définitif et complet :

*A: SQUARO.COM/N
*□

Et c'est fini mais attendez.

d) Exécution.

*/ G et c'est parti pour l'éxécution. N'oubliez pas de répondre aux questions que vous vous étiez vous même posé et à la fin vous revenez en A>.

Surtout attention aux virgules qui séparent les programmes à chainer et aux "/S" qui impose la recherche(Search)des routines nécessaires.

Le signes "/N" impose l'écriture sur le disque désigné de SQUARO.COM le programme définitif objet.

Le "/G" impose l'éxécution. C'est le moment fatidique où il vous indique où il en est. Les BYTES occupés et autres 'salades' et où il dit

[BEGIN EXECUTION]

et c'est parti n'oubliez pas de répondre à la conversation.

3) EXECUTION DIRECTE

Votre SQUARO à marché à l'appel de */G c'est bien. Vous n'avez plus besoin de SQUARO.PRN ni de SQUARO.REL à moins que vous ne prévoyez de linker SQUARO.REL dans un tout autre programme ou dans la librairie LIBREL qui est à votre disposition. Sinon, vous pouvez faire ERA.

Mais il faut faire tourner notre programme objet. C'est simple comme par magie.

A> SQUARO [R] Return Si SQUARO.COM est sur A: et c'est parti. (c'est SQUARO.COM qui tourne)

Je vous souhaite la même chance pour nos XXX.COM.

Pour ma part ça va. Merci.

J'ai plusieurs petites choses pour démonstration.

Mais je suis inquiet pour quand j'entreprendrai du 'Sequentiel Indéxé".

AMSTRAD/SCHNEIDER
FORTH JEDI /AMSCLUB

```
100 LIST
SCR # 100
( TRACE DE SINUSOIDE )
0 VARIABLE AMPLITUDE   180 AMPLITUDE !
0 0 2VARIABLE FREQUENCE   1 3 FREQUENCE 2!
: TRACE
640 0 DO
      AMPLITUDE @  I FREQUENCE 2@ */  *SIN 200 +   I PLOT
LOOP ;
: AXES ( --- )
200 0 PLOT 0 639 DRAWR
;
;S
```

Le programme proposé ici, du nom de TRACE permet de représenter graphiquement le tracé d'une sinusoïde sur l'écran de votre AMSTRAD/SCHNEIDER, ceci en langage FORTH. La formule développée ici en notation polonaise inverse s'inspire de l'expression:

y=200+sin(F*x)*A

où F est la fréquence et A l'amplitude de la courbe.

Comme il n'est pas possible d'utiliser les nombres non-entiers, la fréquence sera représentée par un nombre fractionnaire P/Q; pour cinq cycles, on initialisera la variable double précision avec P=5 et Q=1 ainsi:

5 1 FREQUENCE 2!

puis on demande le tracé de la courbe en tapant TRACE.

Le mot AXES trace l'axe horizontal Ox.

Je vous laisse le soin, à partir de cet exemple, de programmer diverses options telles:

- graduation de l'axes Ox
- tracé de l'axe Oy et sa graduation.
- tracé de courbes multiples.
- sommation de deux ou plusieurs courbes de fréquence et d'amplitude variable.
- modulation d'une courbe par une autre (modulation en anneau).

De plus, l'AMSTRAD possède l'option "tracé en mode XOR". A partir de cette option, tracez une barre verticale dont le déplacement latéral est commandé par les touches fléchées. Les valeurs x et y de la fonction à l'endroit de l'intersection de la courbe avec la barre verticale seront éventuellement affichées.

Et pour finir, si le coeur vous en dit, remplissez un tableau de données avec les différentes valeurs d'une courbe composée et injectez le résultat dans le convertisseur AN/NA dont il est fait mention dans un précédent article de ce numéro.